VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Webové rozhraní pro konfiguraci RaspberryPI a zobrazování dat z Entras API

Web Interface for Raspberry PI Configuration and Entras API

2018 Adam Čech

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Adam Čech**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Webové rozhraní pro konfiguraci RaspberryPI a zobrazování dat z Entras API
Web Interface for Raspberry PI Configuration and Entras API

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je vytvořit webové rozhraní pro konfiguraci RaspberryPI a zobrazování dat z Entras API. Toto rozhraní je postaveno na technologii REST a poskytuje přístup k datům a funkcionalitě projektu Entras, který zajišťuje správu přístupových karet a přístupová oprávnění k místnostem a zónám. Back-end část bude využívat technologie Java.

Konfigurační web RaspberryPI umožní:
1. Změnu hesla uživatele.
2. Nastavení IP adresy - konfigurace DHCP, druhé adresy.
3. Nastavení času a konfiguraci NTP serveru.
4. Zobrazení vytížení, místa na disku.
5. Konfigurace převodníku USR-N540 a obdobných s omezenou množinou možností (jen IP a port - ostatní parametry jsou neměnné).
6. Prozkoumání možnosti otevření TCP socketu a zaslání RAW dat pro Entras čtečku, přímo z prohlížeče.
7. Zaslání RAW dat pro čtečku ze serveru.
8. Web pro přístup k REST API Entras umožní s využitím existujícího REST API Entras zobrazení existujících entit a v omezené míře jejich editaci.

Práce bude obsahovat:
1. Přehled používaných technologií.
2. Automatizované testy funkcionality UI.
3. Programátorskou dokumentaci řešení s využitím diagramů jazyka UML.

Seznam doporučené odborné literatury:

[1] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (Gang of Four): Návrh programů pomocí vzorů. Grada. Praha 2003. ISBN 8024703025
[2] DARWIN, Ian F. Java cookbook. 2nd ed. Sebastopol, CA: O'Reilly, c2004, xxiv, 829 p. ISBN 05-960-0701-9. Dostupné z: http://it-ebooks.info/book/2249/

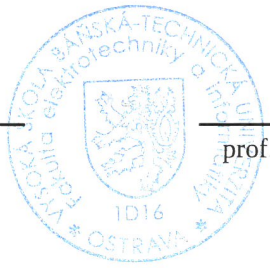
Dále podle pokynů vedoucího práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. David Ježek, Ph.D.**

Datum zadání: 01.09.2017

Datum odevzdání: 30.04.2018



doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. dubna 2018 .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 30. dubna 2018 .....

Na tomto místě bych rád poděkoval panu Ing. Davidovi Ježkovi, Ph.D., za cenné připomínky a rady při vedení této bakalářské práce.

Abstrakt

Tato bakalářská práce se věnuje tvorbě webové aplikace pro konfiguraci Raspberry Pi a správu prostředků využívaných uvnitř přístupového systému Entras. Řeší problémy spojené s přístupem ke službám a zdrojům poskytovaných operačním systémem Raspbian a dále s konfigurací ethernet-serial převodníků pomocí sítě, přístupem ke čtečkám NFC karet skrze tyto převodníky a práci s aplikačním rozhraním přístupového systému Entras.

Klíčová slova: Bakalářská práce, Angular, Spring, Raspberry Pi, Bash, Správa počítače.

Abstract

This bachelor thesis is focused on development of web application for Raspberry Pi configuration and management of resources being used within Entras access system. The thesis solves problems associated with access to services and resources provided by operating system Raspbian and configuration of ethernet-serial converters via the network, access to NFC readers through these converters and work with application interface of Entras access system.

Key Words: Bachelor thesis, Angular, Spring, Raspberry Pi, Bash, Computer management.

Obsah

Seznam použitých zkratek a symbolů

API	–	Application Programming Interface
BASH	–	Bourne-Again SHell
CSS	–	Cascading Style Sheets
CSRF	–	Cross-site Request Forgery
CRUD	–	Create, Read, Update, Delete
HTML	–	Hyper Text Markup Language
HTTP	–	Hypertext Transfer Protocol
HTTPS	–	HTTP Secure
IP	–	Internet Protocol
JS	–	JavaScript
JSON	–	JavaScript Object Notation
LAN	–	Local Area Network
MVC	–	Model-View-Controler
NFC	–	Near Field Communication
UI	–	User Interface
NG	–	Angular
OOP	–	Object-Oriented Programming
OS	–	Operating System
NTP	–	Network Time Protocol
RAM	–	Random-Access memory
REST	–	REpresentational State Transfer
SSH	–	Secure SHell
TCP	–	Transmission Control Protocol
TS	–	Type Script
UDP	–	User Datagram Protocol
UML	–	Unified Modeling Language
WS	–	WebSocket

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Tato bakalářská práce se věnuje tvorbě správcovské webové aplikace pro Raspberry Pi, v rámci přístupového systému Entras. Raspberry Pi se zde nasazuje jako server, v němž se konfiguruje přístupová politika, podle potřeb konkrétního nasazení.

Smyslem aplikace je poskytnout nástroj, který umožní rychlejší a jednoduší konfiguraci Raspberry Pi a služeb, které poskytuje nebo využívá. Práce se zabývá implementací požadavků takového nástroje a řeší problemy s tímto spojené.

Mezi hlavní problémy, které tato práce řeší tudíž patří: přístup aplikace k systémovým zdrojům, konfiguraci již fungujících služeb Raspbianu, práce se síťovým provozem, připojení k ethernet-serial převodníkům pomocí serverové části aplikace a zasílání dat připojeným NFC čtečkám do sériového portu, a přihlášení k již existujícímu REST API Entras pomocí servrové části aplikace a práce s ním.

Práce je zaměřená převážně na praktickou část a kromě těchto praktických úkolů, které řeší, zkoumá i možnosti otevření TCP socketu a zasílání RAW dat pro Entras čtečku, přímo z prohlížeče.

1.1 Obsah kapitol

2. Kapitola popisuje použité technologie.
3. Kapitola uvádí aplikaci do širšího kontextu a požadavků systému Entras.
4. Kapitola představuje návrh a funkcionalitu této aplikace.
5. Kapitola se věnuje řešení a analýze jednotlivých požadavků.
6. Kapitola přináší zhodnocení aplikace a uzavírá práci.

2 Technologie

2.1 REST

REST je klient-server architektura aplikačního rozhraní, která umožňuje přistupovat k datům a provádět nad nimi CRUD operace. REST je bezstavový, čímž jednak značně zjednodušuje komunikaci s API a umožňuje paralelní zpracování obsahu. Zároveň ho lze dost snadno použít s HTTP a v neposlední řadě poskytuje určitý standard, takže není problém použít cizí aplikační rozhraní nebo naopak nabízet vlastní velkému množství dalších uživatelů.

Rozhraní REST je použitelné pro jednotný a snadný přístup ke zdrojům. Všechny zdroje mají vlastní identifikátor URI a lze k nim přistupovat pomocí odpovídajících metod implementovaných protokolem HTTP převážně POST, GET, PUT, DELETE.

Jednou z dalších výhod REST API je uložitelnost do krátkodobé paměti, odpovědi serveru by měly být označeny zda jsou uložitelné do krátkodobé paměti, čímž se urychlí zbytečná komunikace mezi serverem a klientem, pokud se očekává, že se odpověď nebude měnit. [1]

Aplikace vzniklá v rámci této práce má také REST architekturu, kterou poskytuje server Tomcat skrze aplikaci vytvořenou ve Springu a pro přístup k jejímu rozhraní je vytvořená webová aplikace pomocí Angularu.

2.2 TypeScript

TypeScript je relativně nový open-source jazyk, vyvíjený Microsoftem. Jedná se v podstatě o transpilátor do jazyku JavaScript. Důvod, proč se TypeScript jmenuje tak, jak se jmenuje, je že nad JavaScript přidává typový systém, statiku, třídy, interfacy, enumerátory, moduly, apod.

Mezi další výhody, které TypeScript přináší patří i to, že díky typovosti umožňuje odhalit chyby v typech již při vývoji, dělá refactoring bezpečnějším a dbá více na OOP, což dělá vývoj aplikací snadnějším a rychlejším, díky robustnějším a strukturovanějším zdrojovým kódům.

Ukázka jak tento jazyk vypadá je zobrazená v následujícím výpisu 1.

```
1  // Zdrojový kód před kompilací v TypeScriptu
2  class Auto {
3      motor: string;
4      constructor(motor: string) { this.motor = motor; }
5      log(): void { console.log("motor="+this.motor); }
6  }
7  // Zdrojový kód po kompilaci v JavaScriptu
8  var Auto = /** @class */ (function () {
9      function Auto(motor) { this.motor = motor; }
10     Auto.prototype.log = function () { console.log("motor=" + this.motor); };
11     return Auto;
12 }());
```

Výpis 1: Zdrojový kód před a po kompilaci z jazyka TypeScript do jazyka JavaScript.

2.3 Angular

Angular je platforma pro tvorbu front-end částí webových aplikací, která využívá výše zmíněného jazyku TypeScript. Vychází z AngularJS, což je jeho první verze (Angular 1), ale od verze 2+ byl kompletně přepsán a nazývá se pouze Angular. Jedním z největších rozdílů mezi Angular a AngularJS, od kterého se odvíjí i změna v názvu, je kompletní přechod z jazyku JavaScript do jazyku TypeScript.

Jedná se o platformu vyvíjenou Googlem, která je navržená pro snadný vývoj front-end částí moderních webových aplikací. K dosažení tohoto cíle poskytuje knihovny a prostředky, jenž řeší problémy s tímto spojené.

Mezi tyto nástroje patří například knihovna HttpClient, která umožňuje jednoduchou tvorbu HTTP požadavků, ctění MVC architektury, nástroje pro debugování nebo testování jednotlivých komponent aplikace, komprimace zdrojových kódu, či v současnosti snaha cílit na vývoj multiplatformních aplikací.[2]

Angular vytváří tzv. single-page application (SPA). Taková aplikace je tvořená pouze jednou dynamickou stránkou, která obsahuje také všechny podstránky, styly a potřebné skripty. Aplikace tímto umožňuje rychlý a pohodlný přechod mezi jednotlivými podstránkami, bez nutnosti vytvářet nový požadavek a znovunačítat podstránky. Stále je ovšem nutné komunikovat se serverem pro získání konkrétních dat, proto se vytváří v pozadí požadavky na server a data se posílají pomocí XML nebo JSON, to samé poté ze serveru přichází a šablony jsou upravené tak, aby se v nich získané data náležitě zobrazily. Takto vytvořená webová aplikace se přibližuje plynulostí uživatelského rozhraní těm desktopovým. Za zmínku stojí i jedna z dalších výhod, že taková webová stránka může fungovat i offline (pokud ovšem vyžaduje data od serveru, tak daná funkce nebude fungovat). Nevýhodou pak může být to, že se stránka při prvním načítání může načítat trochu déle a zdrojové kódy mohou být trochu komplikovanější. [3] Angular implementuje mechanismus pro tvorbu SPA, skrze knihovnu *Router*, která nabízí odpovídající funkcionalitu pro "routování"mezi podstránkami.

2.4 Spring

Spring je open-source platformou pro tvorbu J2EE aplikací. V podstatě se jedná o kontejner, který umožňuje využívat různé moduly poskytující požadovanou funkcionalitu na základě potřeb dané aplikace, díky čemuž nemusíme znovuvytvářet již vytvořené. [4]

V rámci této aplikace je využit Spring Boot, který je balíkem základních modulů platformy Spring pro tvorbu webových stránek a kontrolérů, poskytuje také možnosti pro jednodušší nasazení. Toho dosahuje například tím, že umožňuje konfigurovat web pouze pomocí anotací a rozšířením speciálních konfiguračních tříd, díky čemuž je možné úplně se vyhnout konfiguraci webu přes .xml soubory, ukázka takové konfigurace je zobrazená na výpise 2.

Mezi základní moduly, které Spring Boot obsahuje, patří například modul pro tvorbu REST kontrolérů, serializaci objektů předávaných kontroléry do JSON formátu nebo Tomcat plugin, který umožňuje spouštět aplikaci, aniž by byl potřeba Tomcat server.

Nejdůležitějším modulem, který bylo potřeba přidat, je jednoznačně modul Spring Security, jelikož se tento modul stará o bezpečnost a ta je v rámci aplikace, která je umístěná uvnitř přístupového systému klíčová. Pomocí tohoto modulu se provádí autentizace a autorizace uživatelů, bez které aplikace nepředá uživateli žádná data nebo nevykoná žádnou činnost, pouze předá přesměrování na přihlašovací stránku a tu následně po vyžádání také odešle. Přístup ke všem kontrolérům je zakázán již v konfiguraci bezpečnostního modulu a jelikož je potřeba mít celou aplikaci zabezpečenou, není potřeba konfiguraci pro přístup ke konkrétním kontrolérům nijak upravovat. V případě potřeby, ale není problém oprávnění pro přístup lehce změnit pomocí přidání anotace *@Secured* k příslušnému kontroléru a následným zadáním povolených rolí, takovou rolí může být i role neautorizovaného uživatele.

Další významnou funkcí bezpečnostního modulu je implementace mechanismu pro ochranu před případnými útoky typu Cross-site Request Forgery nebo Brute-force. Tím odpadá spousta práce, která vyžaduje také hodně odborných znalostí, jediná nevýhoda spočívá v tom, že je nutné spoléhat se na cizí implementaci, ale ta bývá zpravidla prováděná experty na danou problematiku.

Na následujícím výpise 2 je ukázána konfigurace bezpečnostního modulu.

```
1  @Configuration
2  @EnableWebSecurity
3  public class MyWebSecurityConfig extends WebSecurityConfigurerAdapter {
4      @Override
5      protected void configure(HttpSecurity http) throws Exception {
6          http
7              .csrf().csrfTokenRepository(CookieCsrfTokenRepository.withHttpOnlyFalse())
8            .and()
9              .formLogin().loginPage("/").permitAll()
10           .and()
11             .logout()
12             .logoutRequestMatcher(new AntPathRequestMatcher("/logout","POST")).permitAll()
13             .logoutSuccessHandler(new HttpStatusReturningLogoutSuccessHandler())
14             .deleteCookies("JSESSIONID")
15             .invalidateHttpSession(true)
16             .logoutSuccessUrl("/").permitAll()
17           .and()
18             .authorizeRequests()
19             .antMatchers("/index.html","/assets/**","/ng/**","/authentication").permitAll()
20             .anyRequest().authenticated();
21     }
22 }
```

Výpis 2: Konfigurace modulu Spring Security v Spring Boot aplikaci.

2.5 Maven

Maven je systém pro správu a sestavování aplikací postavených nad programovacím jazyce Java. Jeho využitím odpadá závislost na konkrétním vývojovém prostředí, protože všechny informace potřebné ke kompilaci a sestavení programu jsou obsažené přímo ve speciálním souboru pom.xml (POM = Project Object Model).

Repozitář systému Maven obsahuje katalog artefaktů a systém pro jejich vyhledávání a stahování.

Repozitáře jsou lokální a vzdálené. Lokální je přítomen přímo na počítači, který Maven spouští a ukládají se v něm dostupné a již stažené závislosti. Vzdálený se využívá pouze pokud daná závislost není přítomná na lokálním repozitáři a v takovém případě se daná závislost stáhne automaticky z centrálního repozitáře.

Další klíčovou funkcí systému Maven je řešení závislostí. To znamená, že není nutné ručně kopírovat knihovny (JAR, WAR, EAR) a umísťovat je na classpath. Zvláště u velkých projektů složených z mnoha podprojektů je to neocenitelné zpřehlednění a zjednodušení vývoje. [5]

2.6 Apache Tomcat

Tomcat je open-source webový server, vytvořený čistě pomocí progarmovacího jazyku Java, který poskytuje prostředí pro vlastní implementaci webových aplikací pomocí Java technologií. [6] Vzniklá Spring aplikace je určená také pro nasazení na Tomcat.

2.7 Raspberry Pi

Raspberry Pi je miniaturní počítač se slabším výkonem a nízkou cenou, navržený pro nasazení spíše jako rozhraní pro konkrétní aplikace nebo plnění jednodušších úkolů, pro které stačí slabší výkon.

2.7.1 Raspbian

Raspbian je operační systém, který Raspberry Pi využívá. Systém, stejně jako jeho název je odvozen od linuxového OS Debian, akorát je přizpůsoben konkrétně pro potřeby počítače Raspberry Pi.

2.8 Ethernet-Serial převodník

Ethernet-Serial převodník je zařízení, na které se lze připojit pomocí ethernetu a komunikovat se zařízením připojeným do sériového portu. Převodník se používá například v rámci přístupového systému Entras pro konfiguraci čteček NFC karet. V rámci projektu se používají převodníky firmy USR IOT.

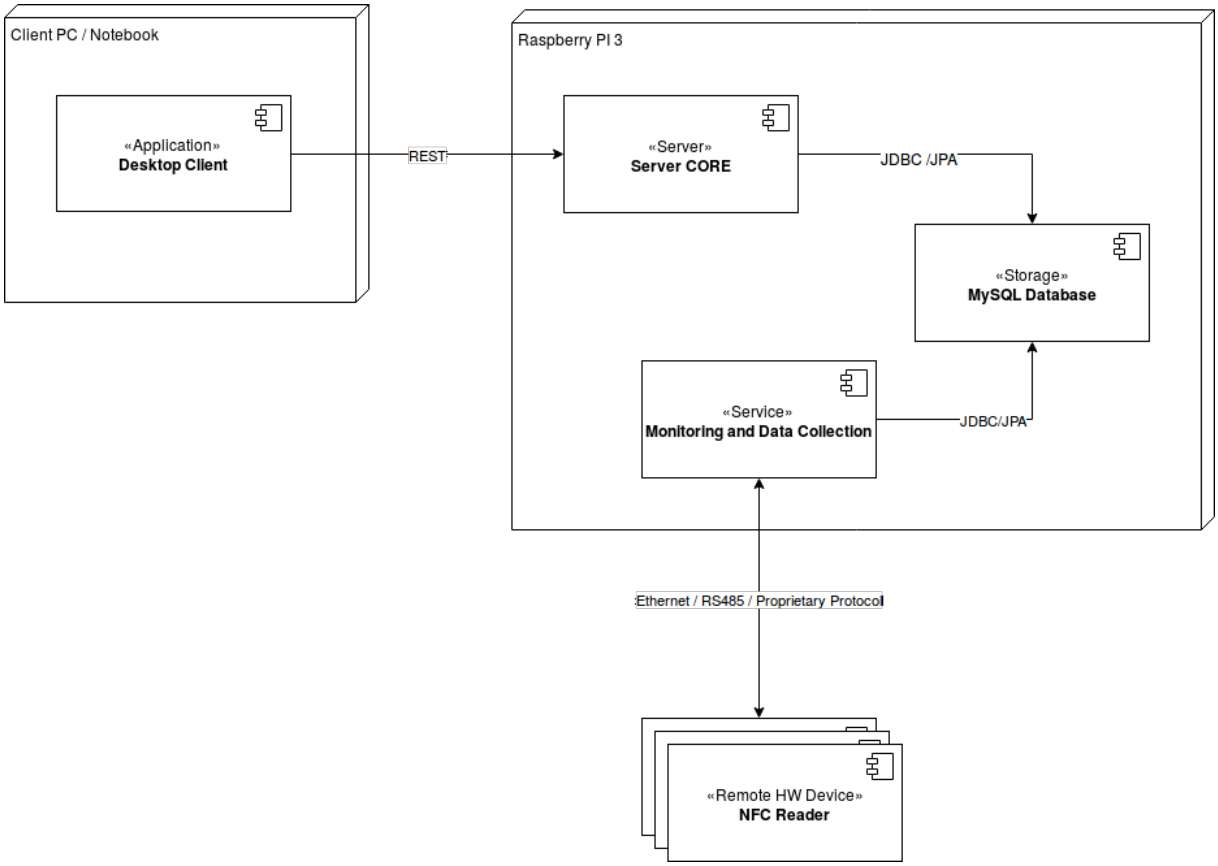
Pomocí TCP protokolu se lze připojit na sériové rozhraní převodníku a komunikovat tímto sériovým portem se zařízením, které je k němu připojené.

Pomocí UDP protokolu lze provádět komunikaci s těmito převodníky v rámci LAN, třeba je vyhledat nebo číst a měnit jejich konfiguraci (např. IP adresace ethernetového portu nebo rychlost přenosu dat sériového portu).

3 Představení přístupového systému Entras

Jedná se o přístupový systém, na jehož vývoji se podílí Vysoká škola báňská – Technická univerzita Ostrava, Katedra informatiky a Moravskoslezský kraj. Přístupový systém je určen například pro zabezpečení škol, veřejných budov, bytových a rodinných domů, či firemních objektů.[7]

Přístupový systém Entras je realizován pomocí **NFC čteček a karet**, jenž ověřují identitu lidí. Obvykle **Raspberry Pi** jako zprostředkovatele přístupové politiky a hlídače systému, jelikož se jedná o levné a dostačující řešení ve spoustě případů, kde se používá menší počet čteček a v případě potřeby může být také lehce vyměněno za výkonnější počítač a dále už jen **Desktopového klienta**, pomocí kterého se konfiguruje přístupová politika. Na obrázku 1 je zobrazené schéma tohoto systému.



Obrázek 1: Celkový pohled na SW systém Entras

Na serveru (Raspberry Pi) jsou umístěné 3 moduly, jenž zajišťují přístupovou politiku:

- **Server Core** - Server, jenž slouží k zpřístupnění potřebných informací o systému.
- **MySQL Database** - Úložiště pro konfiguraci jednotlivých entit daného nasazení.

- **Monitoring and Data Collection** - Služba, která dohlíží na NFC čtečky, jejich funkčnost, konfiguraci, apod.

3.1 Požadavky a umístění aplikace

Aplikace vytvořená v rámci této práce je nasazená na server Tomcat, který je umístěn také na Raspberry Pi. Tato aplikace přímo neovlivňuje ani se nepodílí na funkcionalitě přístupového systému Entras, ale je určena k nastavování systému a zajištění ostatních potřebných funkcionalit při jeho nasazování.

Požadavky pro tuto aplikaci jsou tedy následující:

1. Změna hesla uživatele.
2. Nastavení IP adresy, konfigurace DHCP a záložní adresy.
3. Nastavení času a konfigurace NTP serveru.
4. Zobrazení vytížení a místa na disku.
5. Konfigurace ethernet-serial převodníků firmy USR IOT, pomocí kterých se konfigurují NFC čtečky s omezenou množinou možností (jen IP a port, ostatní parametry např. rychlost přenosu, stop bit, atd. jsou neměnné).
6. Zasílání RAW dat pro čtečky, které jsou připojené k převodníku pomocí serverové části aplikace.
7. Přístup k REST API Entras, zobrazení existujících entit a v omezené míře jejich editaci.

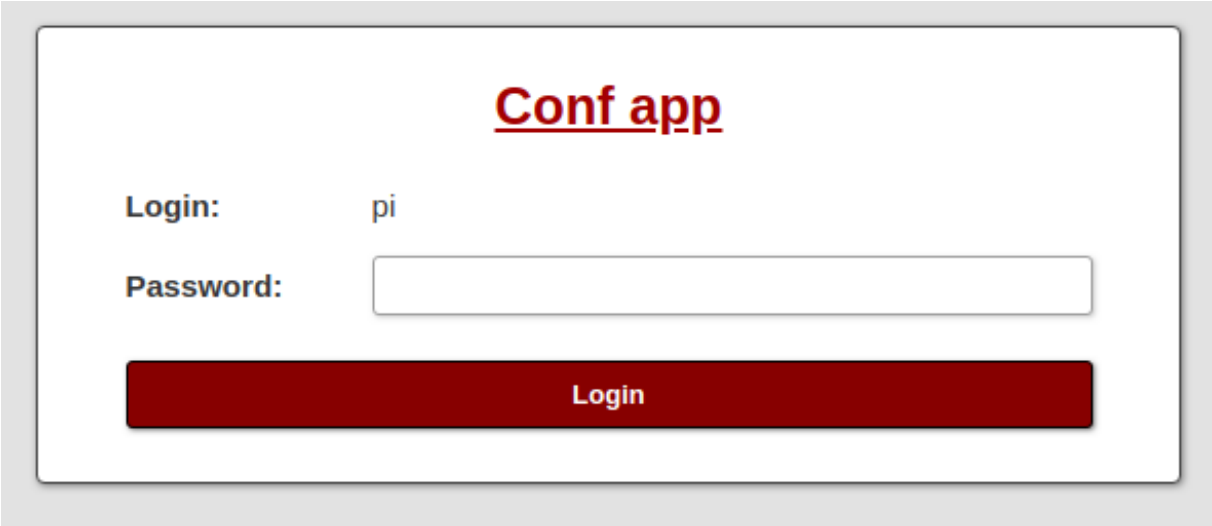
4 Návrh a funkcionalita aplikace

Jelikož je kladen důraz na moderní vývoj, tak aplikace využívá REST architekturu a je vytvořená na základě architekturálního vzoru Model-View-Controller.

Dobré řešení těchto požadavků poskytuje kombinace technologií Angular pro front-end část a Spring pro back-end část. Na straně front-end části se používá i jQuery (JS knihovna) a Font-Awesome (předdefinované CSS styly pro ikony), pro jednodušší tvorbu UI. Pro jednoduché a přenositelné sestavování aplikace se využívá systém Maven. Kromě toho obsahuje aplikace také JUnit testy pro automatizované testování očekávaného chování.

Aplikace vyžaduje také zajištění bezpečnosti, aby neměl přístup k jejím funkcím někdo neoprávněný, proto využívá již zmíněný modul Spring Security, který je podrobněji popsán v kapitole 2.4.

Na obrázcích 2 a 3 je zobrazená ukázka požadovaného vzhledu, pro přiblížení aplikace.



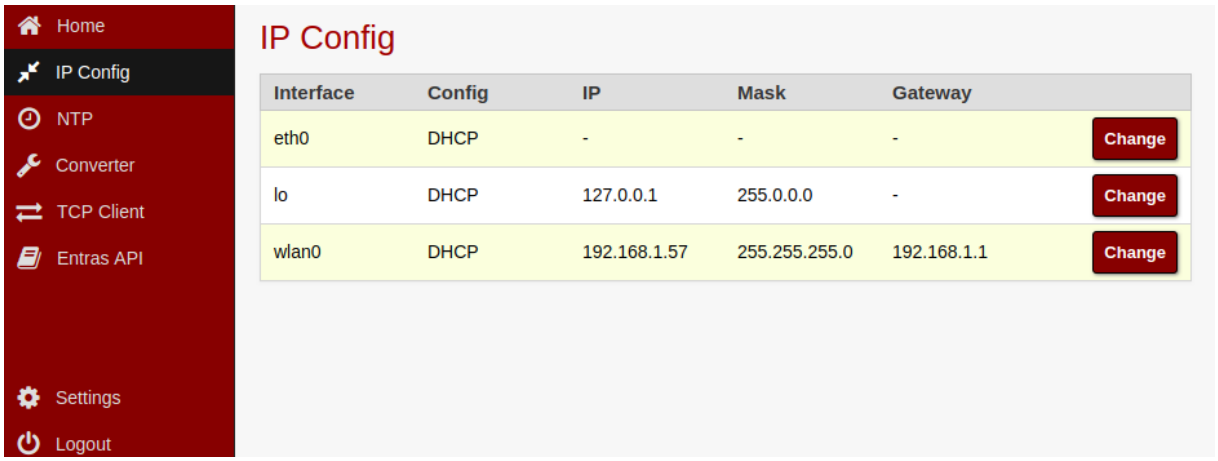
Obrázek 2: Ukázka požadovaného vzhledu přihlašovací stránky

Uživatel se do aplikace přihlašuje pomocí předem daného uživatele *pi*, umístěného přímo na počítači (viz. kapitola 5.1).

Po autentizaci je uživatel přesměrován do zabezpečené části aplikace, která má 7 sekcí, uvnitř kterých se spravuje konkrétní funkce, pro kterou je sekce navržená. Uživatel má po přihlášení plný přístup do aplikace a není ničím omezován.

- Sekce Home umožňuje zobrazit využití úložišť a RAM.
- Sekce IP Config umožňuje konfiguraci IP adresace na daném zařízení.
- Sekce NTP umožňuje nastavit čas pomocí synchronizace NTP nebo HW a SYS hodin, Vypnout/Zapnout/Restartovat NTP službu a přidávat/mazat servery NTP pool.

- Sekce Converter umožňuje vyhledávat a konfigurovat ethernet-serial převodníky uvnitř LAN, kde se zařízení nachází.
- Sekce TCP Client umožňuje připojit se na tyto převodníky a číst nebo zapisovat z NFC čteček, které jsou k nim připojené.
- Sekce Entras API umožňuje připojit se k přístupovému systému Entras, zobrazovat konfiguraci jednotlivých entit, editovat ji a mazat.
- Sekce Setiings umožňuje změnit heslo uživatele.



Obrázek 3: Ukázka požadovaného vzhledu aplikace

Přístupový systém Entras využívá ethernet-serial převodníky firmy USR IOT, které umožňují konfiguraci pomocí sítě, proto je aplikace navržená konkrétně pro tyto převodníky. Kromě toho je aplikace navržená pro počítač Raspberry Pi s operačním systémem Raspbian 4.9, který pracuje se službou *dhcpcd* pro IP konfiguraci a *ntp*, kterou je ovšem potřeba doinstalovat.

5 Řešení a analýza jednotlivých požadavků

Tato kapitola se věnuje řešení a analýze jednotlivých problémů spojených s implementací požadavků této aplikace. Kapitoly 5.1 - 5.4 řeší převážně problémy spojené s práci se systémem. Ta se obvykle provádí pomocí terminálu, do kterého se pomocí příkazového řádku zadávají příkazy, kterými se systém ovládá. Pro často prováděné nebo složitější činnosti se vytváří skripty, které vykonají předem dané činnosti a urychlí, či usnadní prováděnou práci. V případě této práce se vyžaduje totéž. Je vyžadována aplikace, která poskytne rozhraní pro automatizovanou správu.

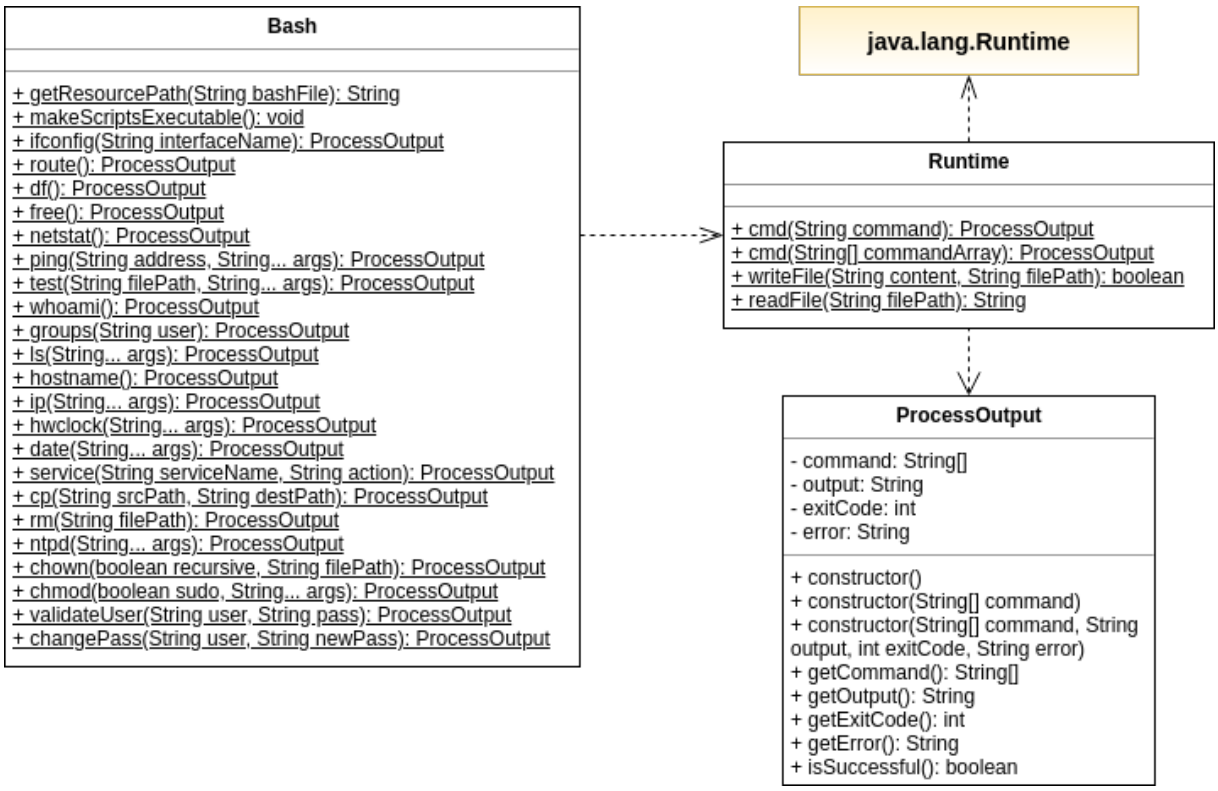
Pro tyto potřeby existuje v Javě třída *java.lang.Runtime*, jenž umožňuje spouštět nové procesy pomocí funkce *exec*, která vrací nový proces jako objekt třídy *java.lang.Process*. Ten již poskytuje objekty typu stream, pomocí kterých lze přečíst výstup spuštěného procesu a dále s ním v aplikaci pracovat. Poskytuje taktéž stream pro vstup, ale ten již není potřeba, jelikož složitější úlohy, které ho potřebují, jsou již řešeny pomocí předpřipravených BASH skriptů, které jsou dělány tak, aby v případě potřeby vstup zadaly a provedly pouze chtěnou činnost či vrátily chtěný výstup. Další užitečnou věcí, kterou třída *Process* poskytuje, je návratový kód spuštěného procesu, díky kterému lze jednoduše zjistit, zda-li byl úspěšný nebo ne.

Ještě je potřeba aby měl uživatel *tomcat* potřebná oprávnění pro spouštění skriptů jako *sudo*, jelikož bez toho by nebylo možné pracovat například se službami nebo přistupovat k systémovým souborům. To však lze udělat v systému Raspbian a linuxových systémech velice jednoduše upravením souboru */etc/sudoers*, který obsahuje bezpečnostní politiku příkazu *sudo*. Přidáním záznamu *tomcat ALL=(root) NOPASSWD: /cesta/ke/skriptu.bash* získá uživatel *tomcat* potřebná oprávnění pro to, aby mohl spouštět zadaný skript jako *sudo*. Stačí zde tedy zadat výčet skriptů vytvořených v rámci aplikace.

Skripty jsou umístěné spolu s aplikací uvnitř *.war* souboru, který se nasazuje na server Tomcat, čímž se stávají lehce přenositelné. Jediný problém, který může nastat, je že nebudou spustitelné, nicméně tento problém zvládne aplikace vyřešit sama, zadáním příkazu *chmod -R u+x /adresar_aplikace/bash* (cesta k adresáři aplikace je lehce zjistitelná pomocí Javy), tento příkaz jim nastaví spustitelnost a jelikož je uživatel *tomcat* zároveň jejich vlastníkem, nepotřebuje ani oprávnění pro spuštění tohoto příkazu jako *sudo*. Pokud tedy v aplikaci dojde k této chybě při spuštění některého ze skriptů vyřeší se sama a skript se opětovně spustí pomocí rekurze, aniž by znovu došlo k této chybě při druhém pokusu.

Aby byla práce se systémem jednoduší, je v aplikaci balík *environment*, který je zobrazen na obrázku 4. Tento balík má vlastní třídu *Runtime*, která využívá funkcí třídy *java.lang.Runtime*, jedná se v podstatě o návrhový vzor Gateway, který pouze zapouzdřuje funkcionalitu. Tento vlastní *Runtime* umožňuje jednoduše spouštět procesy a vracet jejich výstup jako objekt třídy *ProcessOutput*, do kterého se po ukončení procesu uloží všechny potřebné informace a jsou zde již přečtená všechny data z objektů typu stream. Pro další usnadnění obsahuje balík také třídu *Bash*, která poskytuje funkce, jenž už volají konkrétní terminálové příkazy. Třída *Bash* je v podstatě opět Gateway pro *Runtime*. Funkcionalita tohoto balíku tedy umožňuje například

kopírování souboru pouze pomocí zavolání *Bash.cp("/cesta/k/souboru", "/cesta/k/cili");* nebo ověření dostupnosti počítače pomocí příkazu ping zavoláním *Bash.ping("192.168.1.1", "-c1", "-w3").isSuccessful();* (*-c1* pro jeden pokus, *-w3* pro ukončení po 3 sekundách). Další výhodou využití návrhového vzoru Gateway pro třídu *Bash* je zde jednotná implementace pro použité příkazy, čímž odpadá nutnost zadávání pokaždé stejného příkazu do funkce *Runtime.cmd()* a usnadní se tím případný refactoring.



Obrázek 4: Balík environment pro práci se systémem.

Funkce pro spouštění příkazů uvnitř třídy *Runtime* jsou viditelné pouze v rámci balíku, čímž se předchází jejich volání mimo třídu *Bash*. *Runtime* obsahuje také funkce pro jednodušší čtení a zápis do souborů.

5.1 Přihlášení a změna hesla uživatele počítače

Aby nemohl provádět změny v systému někdo neoprávněný, vyžaduje webová aplikace přihlášení. Do aplikace se přihlašuje přímo pomocí uživatele počítače *pi*, který musí být na zařízení (pokud ne, tak přihlášení není možné).

Pro ověření uživatelského jména a hesla je nutné pochopit, jakým způsobem operační systém tyto informace zaznamenává. Většina linuxových systémů, a to včetně systému Raspbian, tyto informace ukládá do souboru /etc/shadow, v tomto souboru se nachází záznam o hesle pro každého uživatele. Tyto záznamy mají následující formát:

adam:6avqYKlds$4m2HSV3uF6IPEYrDNk40/uWUcMchiUP86BZn9X7jpTwQaXqUFWM9MVQ/UxlWy/8TmHADJavPtfDS0Rvb2GiST1:17596:0:99999:7:::

Je v něm uloženo 8 informací oddělených pomocí dvojtečky. Chronologicky se jedná o:

1. Jméno uživatele
2. Pole s heslem, jenž opět obsahuje více informací oddělených pomocí dolaru. Jedná se o 1) Hashovací algoritmus, který má jednu z následujících hodnot podle použitého algoritmu: 1=MD5, 2=Blowfish, 2a=eksBlowfish, 5=SHA-256, 6=SHA-512, 2) Salt a 3) Hash vytvořený pomocí zvoleného algoritmu ze saltu a originálního hesla
3. Počet dnů od poslední změny hesla uložený pomocí Unix formátu.
4. Minimální počet dnů mezi změnou hesla
5. Počet dnů, po kterých bude vynucená změna hesla
6. Počet dnů předem, kdy bude uživatel upozorňován o brzkém vypršení současného hesla
7. Počet dnů, kdy bude účet zablokován po vypršení hesla
8. Počet dnů od zablokování účtu[8]

Ověřování platnosti kombinace uživatelského jména a hesla s existujícími uživateli počítače zajišťuje následující BASH skript, který je zobrazený na výpisu 3, očekává také dva vstupní argumenty, a to již zmíněnou kombinaci uživatelského jména a hesla. Skript si tuto kombinaci uloží a následně z /etc/shadow vyčte informace o hesle spojeném se zadaným uživatelským jménem. Podle zjištěného hashovacího algoritmu, saltu a zadaného hesla vygeneruje pomocí perlu hash a pokud tento nově vygenerovaný hash odpovídá tomu uloženému, tak se skript ukončí s úspěšným stavem a na základě toho aplikace autentizuje uživatele.

```
1  #!/bin/bash
2  USERNAME=$1
3  PASSWD=$2
4
5  id -u $USERNAME > /dev/null
6  if [ $? -ne 0 ]
7  then
8          echo "Not valid"
9          exit 1
10 else
11         ORIGPASS=‘grep -w "$USERNAME" /etc/shadow | cut -d: -f2‘
12         ALGO=‘echo $ORIGPASS | cut -d’$’ -f2‘
13         SALT=‘echo $ORIGPASS | cut -d’$’ -f3‘
14         GENPASS=$(perl -le ’print crypt(’"$PASSWD"’,"\$’"$ALGO"’\$’"$SALT"’\$")’)
15         if [ "$GENPASS" == "$ORIGPASS" ]
16         then
17                 echo "Valid"
18                 exit 0
19         else
20                 echo "Not valid"
21                 exit 1
22         fi
```

```
23 fi
```

Výpis 3: BASH skript pro ověření uživatelského jména a hesla.

Aplikace se mimo jiné nasazuje v rámci projektu Entras na HTTPS server, čímž je lehce a bezpečně zajištěno, že bude všechna komunikace šifrovaná. Díky tomu si lze dovolit posílat zadané heslo z webového rozhraní do kontroléru pomocí prostého textu, který protokol HTTPS zašifruje. Tím odpadá velká spousta práce, jelikož kdyby se prostý text nezašifroval, bylo by nutné, aby nejdřív kontrolér zaslal webové aplikaci informace o hesle zadaného uživatele (typ hashovacího algoritmu a vygenerovaný salt), čímž by se zároveň tyto 2 informace prozradily, poté by bylo nutné, aby webová aplikace uměla vygenerovat hash všemi 5 algoritmy, které Raspbian používá a až poté by šlo bezpečně zaslat vytvořený hash hesla do kontroléru.

Aplikace umožňuje také změnit heslo uživatele, ale pro tuto funkci lze použít již existující systémový příkaz *chpasswd*, který tuto funkci poskytuje.

Jelikož tento příkaz vyžaduje uživatelské vstupy v podobě vybraného uživatele a jeho nového hesla, je opět zabalen do jednoduchého skriptu zobrazenému na výpise 4, kterému stačí tyto údaje zadat jako argumenty a ten poté zavolá příkaz *chpasswd* a zadané argumenty vloží jako uživatelské vstupy.

```
1 #!/bin/bash
2 echo "$1:$2" | chpasswd
```

Výpis 4: BASH skript pro změnu hesla uživatele.

5.2 Zobrazení využití disků a RAM

Jelikož Raspberry Pi nemá vlastní trvalou paměť (vkládá se do něj SD karta, která může mít menší kapacitu) a stejně tak nejpoužívanější současný model *Raspberry Pi 3* má pouze 1 GB operační paměti, je potřeba mít přehled o vytíženosti těchto zdrojů a pro kontrolu jí umožnit zobrazit, aby se předešlo případným problémům s jejich nedostatkem.

Uvítací stránka po přihlášení do webové aplikace proto zobrazuje přehled s vytížením RAM a využitím disků. Tyto informace lze získat relativně jednoduše pro oba tyto zdroje.

Pro získání informací o vytíženosti RAM lze použít již existující příkaz *free --mega*, který zobrazí výpis vytíženosti operačních pamětí (většinou obsahuje kromě skutečné paměti i virtuální odkládací oddíl) a daným přepínačem *--mega* naformátuje výstup tak, aby byl převeden do megabajtů, které asi tak nejvíce odpovídají skutečným možnostem Raspberry Pi.

Získaní informací o využití disků je téměř stejným problémem, jelikož lze opět využít již existující příkaz *df -BMB*. Tento příkaz zobrazí využití disků a opět jej naformátuje do megabajtů pomocí přepínače *-BMB*.

Následně stačí pouze projít výpisy těchto příkazů, vyčíst požadované údaje a uložit je do odpovídajících objektů. Kontrolér je předá uvítací stránce a ta je zobrazí. Uvítací stránka tyto informace vyžaduje každou sekundu a tak jsou stále obnovována a lze je sledovat v reálném čase.

5.3 Konfigurace IP adresace

Při každém novém nasazení Raspberry Pi a také při jeho následném provozu je potřeba zajistit, aby bylo připojené k síti. Následující funkcí, která má usnadnit správu, je proto možnost zobrazování a konfigurace IP adresace.

Tato funkce je v systému Raspbian zajištěná pomocí služby *dhcpcd*, která poskytuje DHCP a DHCPv6 klienta a kromě toho umožňuje i nastavení statické konfigurace síťových rozhraní.

Pro konfiguraci této služby se používá soubor /etc/dhcpcd.conf. Jedná se o inicializační skript, který služba využívá při svém spuštění. Zde je však potřeba vyřešit také problém s přístupovým oprávněním k tomuto souboru, aby byl pro aplikaci přístupný. Vlastníkem je totiž *root* a patří skupině *netdev*, proto k němu *tomcat* nemá přístup, jelikož ostatní mohou soubor pouze číst, ale nemůžou dělat úpravy.

Tento problém by se tedy měl vyřešit již při nasazování aplikace, přidáním uživatele *tomcat* do skupiny *netdev*. Aplikace je však vytvořená tak, aby využila svých *sudo* pravomocí a problém vyřešila sama. Pro čtení souborů používá klasický *java.io.BufferedReader*, pokud však dojde k této výjímce, kdy čtení selže kvůli nedostatku přístupových práv, aplikace překopíruje soubor jako sudo do svého adresáře (umístěného uvnitř adresáře serveru Tomcat), převezme vlastnictví k souboru, opět se jej pokusí přečíst a poté jej smaže. Problém se zápisem je řešen podobně, pokud klasický *java.io.BufferedWriter* selže, tak se data zapíšou do nového souboru v adresáři aplikace a ten se následně překopíruje pomocí sudo pravomocí na místo toho originálního. Poté se ještě smaže nový soubor, který byl vytvořený v adresáři aplikace.

Konfigurace IP adresace se provádí pomocí již zmíněného skriptu */etc/dhcpcd.conf*, ten může obsahovat statickou konfiguraci jednotlivých síťových rozhraní a v případě, že zde rozhraní není zmíněno, tak je pro něj vyžadována adresa od DHCP serverů. Umožňuje také nastavit pro rozhraní záložní statickou adresu, která se použije pouze v případě, kdy rozhraní nezíská žádnou adresu od DHCP serverů.

Mimo konfiguraci síťových rozhraní obsahuje tento skript také možnosti, se kterými má DHCP server volat. Jedná se například o identifikaci pomocí názvu počítače nebo MAC adresy, rezervaci IP adresy i po deaktivaci síťového rozhraní nebo další požadavky pro počítač, které bude klient vyžadovat, to zase mohou být například adresy DNS nebo NTP serverů.

Webová aplikace umožňuje měnit pouze konfiguraci spojenou se síťovými rozhraními a zbytek obsahu uvnitř konfiguračního skriptu nemění, nicméně s ohledem na případné budoucí změny obsahuje aplikace balík s třídami pro jednoduchou práci se síťovými rozhraními a změnami v konfiguraci dhcpcd klienta.

Pro práci s dhcpcd konfigurací je v aplikaci třída *DhcpcdConf*, která projde tento konfigurační soubor a vyčte z něj úplně všechny informace, které uloží do jednoho ze tří odpovídajících seznamů. Tyto seznamy jsou tři a ukládají:

1. Síťové rozhraní reprezentované třídou *NetworkInterface*
2. Známé konfigurační příkazy reprezentované třídou *DhcpcdCommand*

3. Požadavky pro server, neznámé příkazy a komentáře, toto všechno je reprezentované pomocí řetězců.

S konfigurací lze následně pracovat pomocí odpovídajících funkcí, které umí ověřit, zda-li se v konfiguraci něco nachází nebo poskytnout, aktualizovat, či smazat konkrétní objekty. Poskytuje také příslušné funkce pro vygenerování nové konfigurace na základě všech informací, které jsou uložené uvnitř seznamů, a poté ji zapsat.

Pro získání aktuálních informací o síťových rozhraních (jelikož z konfiguračního souboru */etc/dhcpcd.conf* nelze vyčíst například adresu přiřazenou pomocí DHCP serverů) je v aplikaci třída *NetworkInterfaceFactory*. Tato třída poskytuje statické funkce *getAllInterfaces* pro získání informací o všech síťových rozhraních, které jsou na počítači, a *getInterface* pro získání informací pouze o jednom. Aktuální informace se zjistí jednoduše pomocí již existujících příkazů *ifconfig* a *route*, jelikož jde pouze o IP adresu, masku a bránu daného rozhraní.

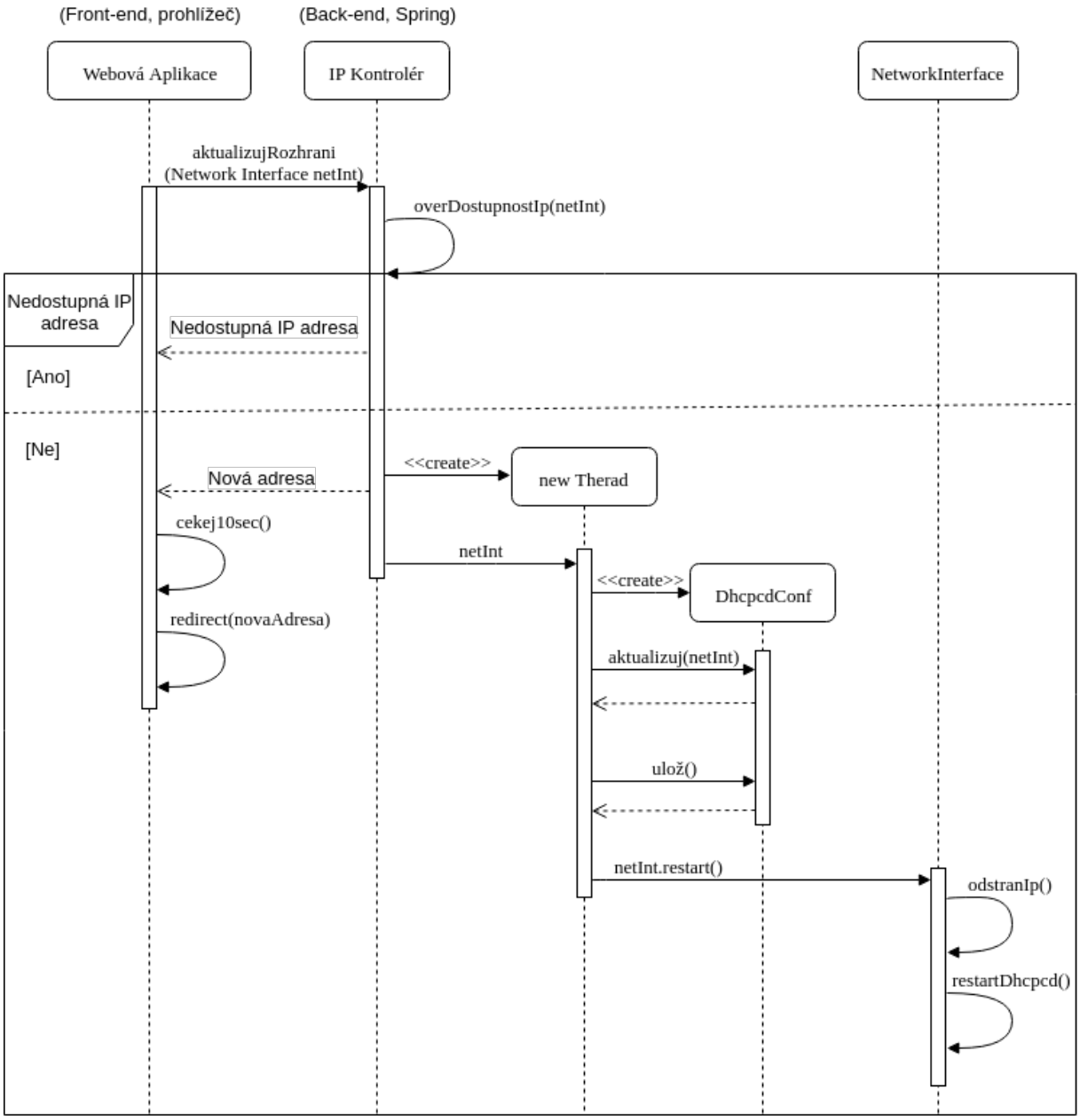
5.3.1 Proces změny konfigurace

Pokud provede uživatel změnu konfigurace některého ze síťových rozhraní pomocí webové aplikace, zavolá se kontrolér, do kterého je tato nová konfigurace předána. Když se v ní nachází statická adresa, ověří se její dostupnost pomocí jednoho pingu a pokud není volná tak aplikace zabrání změně a upozorní uživatele o kolizi. V opačném případě, kdy je IP adresa volná, už aplikaci nebrání nic ve změně a dojde přenastavení.

Ke změně nastavení se používá výše zmíněná třída *DhcpcdConf*, která projde současnou konfiguraci, aktualizuje informace o síťovém rozhraní na základě dat zaslaných webovou aplikací a poté novou konfiguraci trvale zapíše. Aby se změna konfigurace projevila nestačí pouze službu *dhcpcd* restartovat, ale nejdřív je nutné odebrat adresu z konkrétního síťového rozhraní, kde je vyžadována její změna, jelikož *dhcpcd* klient ignoruje rozhraní, které už jsou nakonfigurované. Nejdřív je tedy nutné z daného rozhraní odebrat adresu a toho lze docílit jednoduše pomocí příkazu *ip addr flush dev [název_rozhraní]* a poté již stačí službu *dhcpcd* restartovat a síťové rozhraní získá novou adresu. Tento proces je ovšem potřeba provést v odděleném vlákně, jelikož kontrolér musí odpovědět webové aplikaci dříve než odstraní IP adresu ze síťového rozhraní, čímž způsobí přerušení spojení.

Webová aplikace počká po odpovědi kontroléru deset sekund, během kterých by měl server získat novou IP adresu, a poté provede přesměrování na základě zaslané konfigurace. V případě nastavení statické IP adresy je to snažší, jelikož webová aplikace zná novou adresu a může uživatele přesměrovat přímo na ni. V případě DHCP a fallbacku (záložní adresy) je to složitější, jelikož IP adresu přiřadí DHCP server a webová aplikace tuto adresu dopředu nezná. V takovém případě kontrolér přidá do své odpovědi název počítače a jelikož si jej DHCP zaregistruje pod tímto názvem při zapůjčení IP adresy, provede se přesměrování na tento zaslaný název počítače.

Na následujícím obrázku 5 je detailně zobrazeno jak se celý systém chová během procesu změny IP adresy.



Obrázek 5: Chování systému při změně IP adresy

5.4 Konfigurace NTP služby

V rámci přístupového systému Entras je potřeba, aby měly všechny zařízení stejně nastavený čas a bylo možné přesně zaznamenávat všechny události vyvolané uvnitř systému, proto je zde NTP služba klíčová k tomu, aby zajistila přesný a synchronizovaný čas.

U konfigurace NTP se jedná o podobné problémy jako ty, které řeší předchozí kapitola. Je zde řešena práce se službou *ntp*, kterou je potřeba doinstalovat, a synchronizace času po-

mocí NTP klienta nebo přístupu k hardwarovým hodinám a jejich vzájemné synchronizaci se systémovými hodinami.

Raspberry Pi bohužel nemá vlastní hardwarové hodiny, které by zachovaly aktuální čas i po jeho vypnutí, ale lehce se dají k jeho základové desce připojit. Pro spoustu potřeb stačí pouze systémové hodiny, které se synchronizují již při spuštění počítače pomocí NTP klienta pokud má přístup k NTP serveru a tak si uživatel častokrát ani nevšimne špatného času, nicméně v případě nasazení Raspberry Pi v přístupovém systému Entras je aktuální čas velice důležitý a tak se k němu hardwarové hodiny připojují, proto bylo potřeba vytvořit funkce i pro jejich vzájemnou synchronizaci se systémovými hodinami.

Synchronizace hardwarových a systémových hodin se dá opět vyřešit jednoduše pomocí již existujícího příkazu *hwclock*, který umožňuje přistupovat k hardwarovým hodinám a následně pomocí přepínače *--hctosys* nastavit čas na systémových hodinách podle času hardwarových a přepínačem *--systohc* opačně tzn. nastavit čas na hardwarových hodinách podle systémových.

Jelikož ale může být čas na hardwarových nebo systémových hodinách špatně nastavený nebo je vyžadován ještě přesnější čas, lze jej synchronizovat pomocí NTP serverů, proto umožňuje aplikace využít NTP klienta. Pro jeho využití je potřeba nejdřív vypnout *ntp* službu, jelikož využívá stejné porty jako NTP klient, a poté až lze provést synchronizaci času, na závěr je ještě potřeba *ntp* službu opět zapnout. Po dobu synchronizace, která většinou trvá něco málo přes 10 sekund, je tedy služba *ntp* nedostupná. Synchronizace času se provádí pomocí příkazu *ntpd -q*. Jedná se o NTP Daemona, který toto umožňuje, a pomocí přepínače *-q* ukončí svůj proces po prvním nastavení času a tím opět uvolní prostor pro spuštění *ntp* služby.

Aplikace umožňuje také spravovat tuto *ntp* službu. Základními funkcemi, které umožňuje se službou provádět, je její spuštění, vypnutí, restartování a zobrazení stavu pomocí klasického příkazu pro přístup ke službám *service ntp [start|stop|restart|status]*.

Dále poskytuje možnost editace cizích NTP serverů využívaných pro synchronizaci času. Ta se provádí opět v rámci konfiguračního skriptu služby *ntp*. Tímto skriptem je soubor */etc/ntp.conf* (zde je potřeba vyřešit opět stejný problém jako v případě konfiguračního skriptu */etc/dhcpcd.conf* s přístupem k souboru, jelikož patří do skupiny root a vlastníkem je také root). Opět se jedná o konfigurační skript, který je v aplikaci reprezentován příslušnou třídou *NtpConf*, jenž umožňuje zpracovat tento konfigurační skript a jednoduše s ním pracovat. Poskytuje funkce pro přístup k příkazům uvnitř tohoto souboru, jejich editaci a vygenerování nové konfigurace. Kromě seznamu s konfiguračními příkazy obsahuje také seznam s využívanými NTP servery reprezentovanými pomocí třídy *NtpConfPool*.

5.5 Konfigurace Ethernet-Serial převodníků

Základním prvkem uvnitř přístupových systémů jsou zařízení pro ověřování identity lidí pohybujících se uvnitř střeženého objektu. Jednou z nejpoužívanějších technologií, které toto umožňují jsou bezdrátové NFC čtečky, jenž jsou využívané také v případě přístupového systému Entras. Tyto čtečky mají malé rozměry a obsahují pouze nutné součásti pro jejich funkčnost, proto

je možné komunikovat s nimi pouze pomocí sériového portu. Aby bylo možné připojit je do počítačové sítě a komunikovat s nimi na dálku, využívají se ethernet-seriál převodníky. Tyto převodníky se připojují pomocí ethernetu do sítě a zároveň pomocí sériového portu ke čtečce, čímž se tedy převodník stává zprostředkovatelem, který umožňuje pracovat se čtečkou na dálku. Převodníky se tedy umísťují spolu s NFC čtečkami a tím umožňují monitorovat jejich stav, protože je důležité vědět, zda-li jsou v provozu a provádět jejich konfiguraci na dálku. Požadavkem řešeným v této kapitole je tedy to, aby aplikace umožňovala spravovat převodníky a v následující kapitole 5.6 se pak řeší rychlá správa čteček.

O převodnících je nutné mít přehled, proto aplikace umožňuje jejich vyhledávání a provádění konfigurace na dálku. V rámci přístupového systému Entras se proto používají převodníky firmy USR IOT, které umožňují provádět správu pomocí sítě.

Většina konfigurace pro tyto převodníky je pokaždé stejná, jedná se například o rychlost komunikace sériového portu, paritu nebo pracovní mód jako TCP Server. Tento mód umožňuje připojit se na převodník pomocí TCP protokolu a využívat sériový port, podobně jako kdyby byl umístěn přímo na počítači, který je k převodníku připojen. Protože tato konfigurace odpovídá možnostem NFC čtečky, je pokaždé stejná. Konfiguruje se tedy pouze číslo TCP portu, pomocí kterého se lze připojit TCP protokolem na sériový port převodníku, a nastavení statické IP adresy nebo případně DHCP pro síťové rozhraní převodníku.

Konfigurace převodníku se provádí pomocí UDP protokolu na zdrojovém i cílovém portu 1901 a daty zasílanými na broadcast tzn. 255.255.255.255, nicméně novější převodníky poskytují více konfiguračních možností a kvůli tomu využívají i odlišný port 1500. Tyto novější převodníky pouze rozšiřují funkce poskytované těmi staršími a tudíž se základní konfigurace provádí úplně stejně pomocí obou dvou portů, ale kvůli této odchylce je potřeba, aby serverová část aplikace pokryla oba tyto porty.

Konfigurační protokol těchto převodníků je umožňuje vyhledávat, restartovat, zobrazovat jejich konfiguraci, editovat jejich konfiguraci a ukládat ji. Struktura tohoto konfiguračního protokolu je zobrazena v následující tabulce 1.

Funkce	Hlavička	Délka	Id	MAC	Login/Heslo	Parametr	Kontrolní součet
Vyhledávání	FF	01	01				02
Restart	FF	13	02	[MAC]	[Login] [Heslo]		xx
Čtení konfigurace	FF	13	03	[MAC]	[Login] [Heslo]		xx
Uložení konfigurace	FF	13	04	[MAC]	[Login] [Heslo]		xx
Základní nastavení	FF	56	05	[MAC]	[Login] [Heslo]	Nastavení	xx
Nastavení portu 1	FF	52	06	[MAC]	[Login] [Heslo]	Nastavení	xx
Nastavení portu 2	FF	52	07	[MAC]	[Login] [Heslo]	Nastavení	xx
Nastavení portu 3	FF	52	08	[MAC]	[Login] [Heslo]	Nastavení	xx
Nastavení portu 4	FF	52	0F	[MAC]	[Login] [Heslo]	Nastavení	xx

Tabulka 1: USR-TCP232-E45/M4 konfigurační protokol.[9]

Tento protokol obsahuje následující pole:

- **Hlavičku**, která je vždycky stejná
- **Délku**, která zaznamenává počet bajtů od id (včetně) po parametr (včetně)
- **Id** pro identifikaci zaslaného příkazu
- **MAC adresu** pro identifikaci převodníku, kterému je příkaz zaslán (pokaždé 6 bajtů)
- **Login a Heslo** pro ověření uživatele (pokaždé 6B a 6B)
- **Parametr**, do kterého se vkládá celá konfigurace zařízení, ta obsahuje 67B a je v ní např. IP adresace, název zařízení, login, heslo, apod. nebo konfigurace sériového portu, která obsahuje 63B a je v ní např. číslo portu pro připojení přes TCP nebo stop bit.
- **Kontrolní součet**, jenž je bajt vytvořený ze součtu všech polí od délky (včetně) po parametr (včetně).

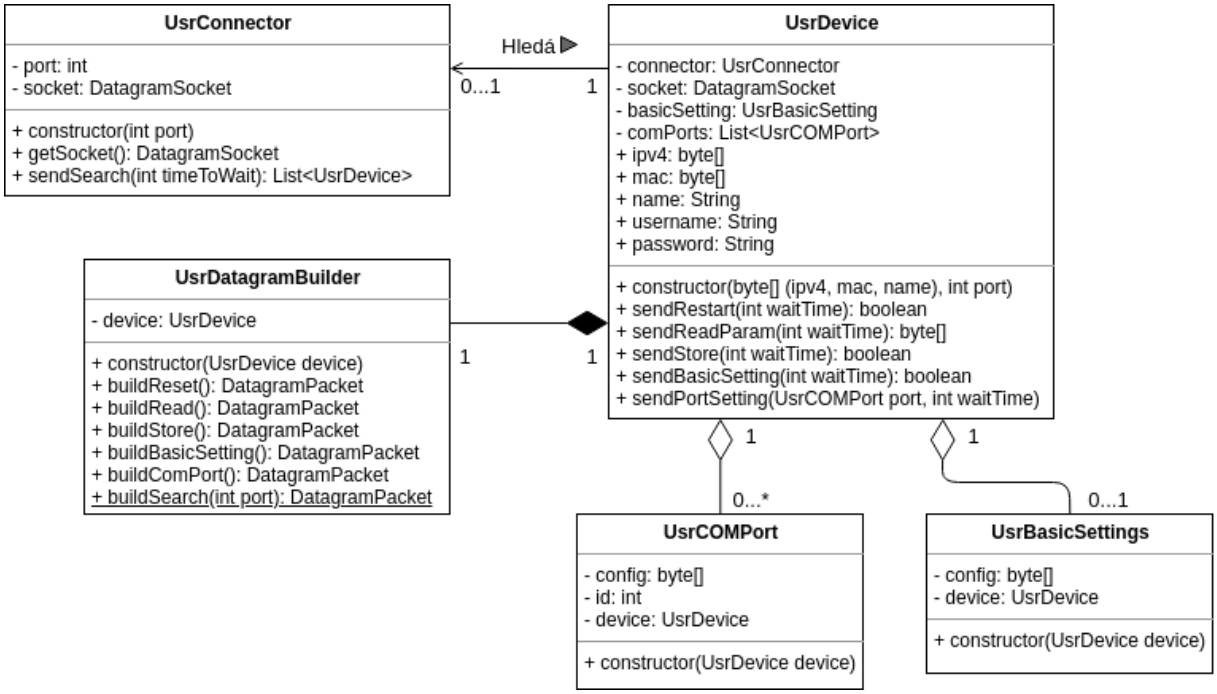
Nejjednodušším požadavkem pro implementaci je vyhledávání převodníků, jelikož je tento příkaz vždycky stejný. Stačí tudíž otevřít socket, zaslat příkaz a počkat na odpovědi jednotlivých převodníků, které se ozvou. Jejich odpověď obsahuje název zařízení, IP adresu a MAC adresu. Na základě toho lze převodníky identifikovat a pracovat s nimi dále.

Následně je vyžadována také správa těchto převodníků, a tak poté, co je aplikace najde, umožní jejich konfiguraci. Poskytuje možnost nastavení statické IP adresy nebo využití DHCP a nastavení čísel TCP portů pro připojení k těm sériovým. Také je potřeba nakonfigurovat převodník tak, aby umožňoval komunikaci s NFC čtečkami. To vyžaduje nastavení rychlosti přenosu jako 9600 bps, velikosti bloku jako 8b, žádnou paritu, 1 stop bit a pracovní mód jako TCP server. Jelikož je ale tato konfigurace neměnná, tak se pokaždé nastaví automaticky a posílá se spolu s ostatním nastavením, aby si ji uživatel nemusel pamatovat nebo ji nenastavil špatně.

Konfigurace převodníků se zjišťuje pomocí čtvrtého příkazu pro její přečtení. Ten je vyvolán automaticky pomocí techniky lazy load, poté co aplikace nalezne převodník, uloží jej do odpovídajícího objektu, který ho reprezentuje, a následně během jeho předávání z REST kontroléru do webového rozhraní se nastaví objekt, který reprezentuje konfiguraci zařízení a konfiguraci portů. Jelikož je aplikace z pouhého vyhledání nezná, tak obsahují pouze prázdný ukazatel a tím se spustí celý proces techniky lazy load. Aplikace vytvoří nový datagram s čtvrtým příkazem, zjistí informace o konfiguraci a poté už je pouze předá do webového rozhraní.

Poté, co uživatel provede změnu nastavení, server pošle převodníku pomocí příkazů zobrazených v tabulce 1 jednotlivé části konfigurace. První se zašle konfigurace zařízení a poté konfigurace pro každý sériový port. Aplikace počká na potvrzení převodníku o přijetí všech nových konfigurací a poté zašle příkaz pro trvalé uložení a následně restartování.

Pro snadnou práci s převodníky je v aplikaci navržený balík *usrtcp232*, který je zobrazen na obrázku 6. Ten poskytuje odpovídající rozhraní, které převodníky reprezentuje a implementuje funkce síťového protokolu pro jejich konfiguraci.



Obrázek 6: Balík usrtcp232 pro práci s převodníky firmy USR IOT.

Základem tohoto balíku je třída *UsrConnector*, která umožňuje vyhledávat převodníky pomocí funkce *sendSearch* a následně s nimi pracovat. V konstruktoru této třídy lze také určit, na kterém portu se má otevřít spojení a tím lze jednoduše vyřešit problém s odlišností starších a novějších převodníků pomocí otevření spojení na obou portech.

Převodníky jsou následně reprezentovány pomocí třídy *UsrDevice*, která obsahuje jejich vlastnosti a umožňuje jim zasílat příkazy. K tomu slouží pomocná třída *UsrDatagramBuilder*, jenž umožňuje jednoduše vytvářet datagramy s odpovídajícími příkazy.

Pomocí tříd *UsrCOMPort* a *UsrBasicSettings* je reprezentována konfigurace převodníku. Ta se v obou třídách ukládá v proměnné *config*, jenž je polem bajtů. Tyto třídy obsahují také více než deset metod typu get/set (jenž nejsou zaznačené v zobrazeném diagramu), které z proměnné *config* vytáhnou, či změní určitou část pole, jenž odpovídá zvolené možnosti konfigurace.

5.6 Zaslání RAW dat pro čtečku ze serveru

NFC čtečky uvnitř přístupového systému Entras obsahují základní vlastnosti jako jsou název, id, verze, čas, časová zóna, apod., a pak také danou konfiguraci konkrétního nasazení, která je uložena ve formátu JSON a obsahuje například informace o uživatelích, skupinách nebo pravomocích. Za účelem rychlé kontroly, případně rychlé úpravy dat uložených uvnitř NFC čtečky, je potřeba, aby aplikace umožňovala přípojení na sériový port převodníku a možnost čtení nebo zápisu dat z připojené čtečky.

Toto je vyřešeno pomocí již vytvořených tříd v rámci projektu Entras pro snadnou práci se čtečkami, kde stačí zadat pouze adresu nebo název převodníku a port, pomocí kterého se má

otevřít spojení. Následně lze pomocí příslušných funkcí pro čtení a zápis přistupovat k datům uvnitř čtečky. Pro přístup k těmto datům je nutné zvolit konkrétní příkaz, kterých je uvnitř čtečky kolem sedmi, a ten načte nebo zapíše část konfigurace (případně konfiguraci celou), kterou příkaz označuje. V případě čtení, se data zapíší do souboru a ten je následně po částech zaslán do aplikace. V případě zápisu, jsou data zapisována do čtečky po částech. Rychlost je zde výrazně omezena možnostmi sériového portu a tak může tento proces trvat i několik minut.

Takové řešení ovšem vyžaduje Java server, pomocí kterého lze oproti prohlížeči vytvořit TCP socket pro spojení s převodníkem. Tím se stává server zprostředkovatelem komunikace a to se dá považovat za nevýhodu, jelikož se tím stává server nutností a zároveň se jedná o jeho další vytížení, a proto se následující kapitola věnuje tomu, co by bylo potřeba udělat, aby nutnost tohoto zprostředkovatele odpadla a prozkoumává možnosti otevření TCP socketu již v prohlížeči.

5.7 Prozkoumání možnosti otevření TCP socketu

Otevření TCP či UDP socketu v prohlížeči momentálně není možné jednoduše provést, jelikož žádný z prohlížečů neposkytuje odpovídající rozhraní, které by tuto funkci implementovalo. Tento fakt je zapříčeněn tím, že by tato funkce mohla ohrožovat samotné uživatele a poskytovala by podvodným webovým stránkám příležitost, jak je zneužít, jelikož by porušovala bezpečnostní politiku stejného původu. Ta zajišťuje, že jakýkoliv požadavek z webové stránky je zaslán stejné aplikaci jako je ta, která požadavek vytvořila, tudíž všechny požadavky webové stránky musí mít stejný protokol, adresu a port, což by otevírání socketů porušovalo.[10]

Existují ale alternativy a experimentální rozhraní, které toto umožňují. V rámci organizace W3C existuje skupina s názvem System Applications Working Group, která se zabývá tvorbou rozhraní pro webové aplikace, které potřebují integraci systémových funkcí jako je práce se sockety, bluetooth, telefonním vytáčením nebo posíláním zpráv.[11]

V roce 2013 bylo zveřejněno touto skupinou rozhraní pro práci se sockety skrze prohlížeč. Jedná se o čisté rozhraní navržené pro budoucí snadnou implementaci funkcionality socketů do prohlížeče, kterou se W3C nikdy nerozhodlo implementovat. Rozhraní popisuje třídy *TCPSocket* a *UDPSocket* pro inicializaci socketu a příslušné funkce pro posílání a čtení dat, obdobně jako v ostatních jazycích, které toto umožňují. Mimo to popisuje také různé typy výjímek a vyžádání povolení od uživtele při otevírání socketu pro danou adresu a port. Skupina se ale po čase rozhodla od tohoto projektu upustit a jeho implementace je momentálně v nedohlednu.[12]

Společnost Mozilla implementovala rozhraní umožňující práci se sockety, ale toto rozhraní je dostupné pouze pro Firefox OS a privilegované nebo certifikované aplikace. Firefox OS je linuxový operační systém postavený na webových technologiích (HTML, CSS, JS), který ale není společností Mozilla nadále podporován (od roku 2016). Toto rozhraní je dostupné skrze zabudovanou vlastnost v prohlížeči *navigator.mozTCPSocket*, která je již objektem třídy *TCP-Socket*. Rozhraní je určené především na tvorbu aplikací pro Firefox OS a jeho implementace do klasického prohlížeče se také nechystá.[13]

Jelikož klasický TCP nebo UDP socket není vhodný pro použití uvnitř prohlížeče, lze použít jako alternativu tzv. WebSocket. WebSocket je obdobou těchto technologií, přenesenou do prostředí webových aplikací. Umožňuje vytvořit obousměrný komunikační kanál (pomocí TCP protokolu), kterým lze komunikovat se serverem v reálném čase.[14] V případě potřeb přístupového systému Entras ale takové řešení není možné, jelikož převodník, se kterým je potřeba navázat spojení skrze TCP socket, neposkytuje žádnou alternativu v podobě webového serveru, který by umožnil vytvořit spojení skrze WebSocket.

5.8 Přístup k REST API Entras

Poslední požadavek, který aplikace vyžaduje se netýká konfigurace žádného zařízení, ale tvorbě rozhraní pro rychlý přístup k Entras API. Toto API se nasazuje také na Raspberry Pi, společně s touto aplikací. Projekt Entras již obsahuje desktopovou aplikaci pro správu přístupové politiky, a tak je cílem této aplikace poskytnout pouze jednodušší rozhraní, které umožní přístup k API pouze pomocí prohlížeče.

Entras API poskytuje REST rozhraní, pomocí kterého lze přistupovat k jednotlivým entitám, které tvoří přístupovou politiku daného nasazení. Tato aplikace umožňuje připojit se k tomuto rozhraní a zobrazovat, editovat nebo mazat jednotlivé entity. Mezi jednotlivé entity, které systém obsahuje, patří například uživatelé Entras API, uživatelé přístupových karet, kteří se uvnitř daného nasazení pohybují, skupiny, pravidla, apod. Data jsou zde předávány nebo vkládány jednoduchým způsobem pomocí JSON notace.

Pro přístup k Entras API vytváří webové aplikace požadavky na serverovou část aplikace a ta následně na REST rozhraní Entras API. To se dá považovat za trochu nestandardní způsob a do jisté míry jako nevýhoda, protože se tím komunikace prodlužuje, jelikož musí jít také přes server, ale odpadává tím problém přihlášení do Entras API, které se provádí pomocí vyplnění HTML formuláře, čímž se stává implementace pomocí Javy a servrové části aplikace vhodnějším a jednodušším. Poté lze již přistupovat k datům klasicky pomocí jejich URI identifikátorů a odpovídajících HTTP metod.

Pro připojení k Entras API je nutné vyplnit formulář, ve kterém se vyplní adresa nebo název počítače, na jehož API se má aplikace připojit (ve většině případů půjde o výchozí *localhost*, nicméně to není podmínkou), a dále je nutné zadat i uživatelské jméno a heslo evidované v daném API. Pokud se povede úspěšně ověřit uživatele, zobrazí se na webové stránce seznam entit a uživatel si může vybrat konkrétní data, které ho zajímají.

Jelikož má většina entit minimálně deset atributů, umožňuje aplikace přepínat mezi seznamovým a tabulkovým vzhledem. Tabulkový vzhled je vhodnější pro entity s menším počtem atributů, ale někdy může být příliš široký a v takovém případě může být vhodnější seznamový vzhled, který zobrazuje záznamy po jednom. Jelikož některé entity můžou mít klidně desítky tisíc záznamů, obsahuje aplikace také stránkování a záznamy se zobrazují po padesáti.

Názvy atributů jednotlivých entit jsou překládány pomocí souboru s lokalizací. V tomto souboru je zapsán objekt ve formátu *JSON*, ve kterém jsou odpovídající názvy atributů přeložené

tak, aby byly srozumitelné pro uživatele. Import JSON souboru lze provést v platformě Angular a jazyce TypeScriptu provést velice jednoduše následujícím způsobem. V prvním kroku, zobrazeném ve výpise 5, je nutné vytvořit modul pro import .json souborů.

```
1  declare module "*.json" {
2    const value: any;
3    export default value;
4  }
```

Výpis 5: Modul pro import JSON souborů do TypeScriptu.

Poté už stačí soubor pouze jednoduše importovat, jak je to zobrazeno ve výpisu 6 na prvním řádku. K hodnotám uvnitř tohoto souboru lze poté přistupovat pomocí aliasu zadaného při importu, tak jak je zobrazeno na řádcích osm a devět.

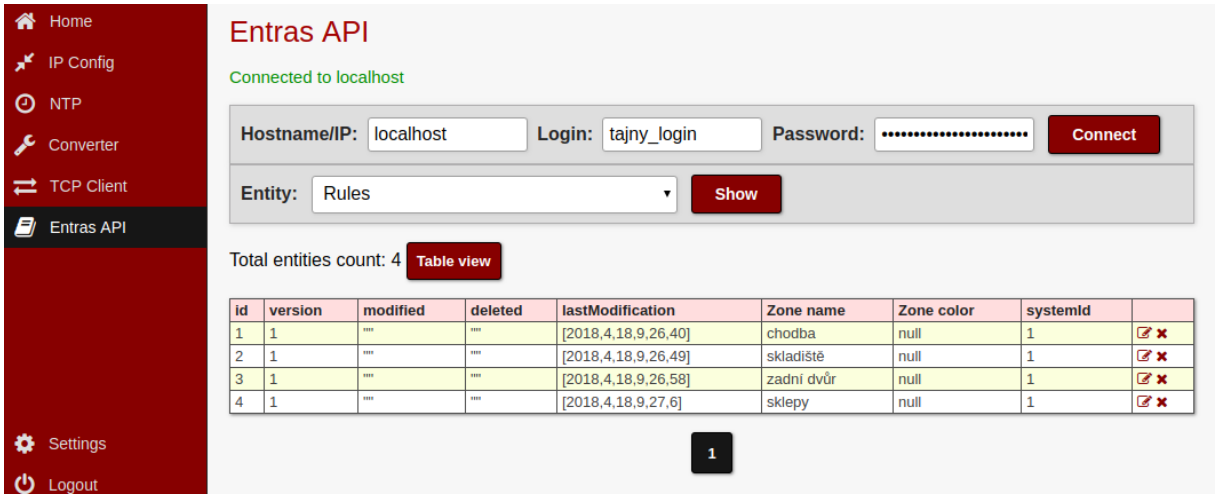
```
1  import * as translations from "../EntrasTranslationsEN.json";
2
3  export class EntrasAPIEntityAttribute {
4    /*...*/
5    constructor (entityName: string, key: string, value: any) {
6      this.key = key;
7      this.value = value;
8      if (transltions[entityName][key] != null) {
9        this.translatedKey = transltions[entityName][key];
10     }
11   }
12
13   get translatedKey() {
14     return this._translatedKey == null ? this.key : this._translatedKey;
15   }
16   /*...*/
17 }
```

Výpis 6: Import JSON do TypeScriptu.

Jelikož se může stát, že některý z atributů nebude mít překlad, nezbývá než použít v takovém případě název z databáze. Toto lze ovšem vyřešit velice jednoduše pomocí metody typu get, jak je zobrazeno ve výpisu 6 na řádku 13.

5.8.1 Práce s Entras API

Na následujícím obrázku 7 je zobrazena ukázka práce s aplikačním rozhraním Entras a v následující tabulce 2 jsou zobrazeny jednotlivé funkce poskytované aplikačním rozhraním Entras, které aplikace využívá.



Obrázek 7: Ukázka práce s aplikačním rozhraním Entras

	HTTP metoda	**URI**	**Operace**
1	GET	/{entityName}	Získání seznamu záznamů
2	GET	/{entityName}/{id}	Získání jednoho záznamu
3	PUT	/{entityName}	Aktualizace záznamu
4	DELETE	/{entityName}/{id}	Smazání jednoho záznamu

Tabulka 2: Funkce poskytované REST aplikačním rozhraním Entras.

Pomocí proměnné *entityName* v cestě požadavku se pokaždé identifikuje daná entita a v případě, že obsahuje i *id* se zároveň identifikuje také konkrétní záznam. První funkcí pro získání seznamu záznamů konkrétní entity, lze zadat také nepovinné parametry pro stránkování, jako počet záznamů a zvolenou stránku. Třetí funkce pro aktualizaci záznamu očekává vložení celého aktualizovaného záznamu do těla požadavku ve formátu JSON.

Práce s tímto rozhraním začíná ve webové aplikaci zvolením dané entity a zavoláním funkce pro získání seznamu záznamů. Uživateli se poté zobrazí již zmíněný tabulkový nebo seznamový výpis podle jeho volby a následně může zobrazené záznamy editovat nebo mazat.

V případě funkce pro odstranění je to jednodušší, jelikož stačí vytvořit HTTP požadavek metody DELETE a zadat do URI název entity a id záznamu. Jediná komplikace, která může nastat je, že záznam nebude možné odstranit, například z důvodu provázanosti s jiným záznamem nebo nedostatku přístupových práv. V takovém případě už ovšem nejde nic dělat a stačí informovat uživatele o neúspěchu.

O něco složitější je to v případě editace. Aplikace vyžádá znovu na základě *id* daný záznam, čímž získá jeho nejaktuálnější podobu, a zobrazí formulář pro editaci jednotlivých atributů. Také je potřeba zachovat stejný JSON formát jako ten, ve kterém záznam přišel, aby jej rozhraní Entras API po změně hodnot přijalo, proto je potřeba dát si pozor při vkládání lokalizace, aby

se nezměnil název některého z atributů a v případě, že je hodnotou atributu nějaký objekt, tak je potřeba převést jej do JSON řetězce a následně před uložením zpátky na objekt.

Přijatý záznam se proto uloží přesně tak jak přišel a poté ještě jednou do pole objektů, které reprezentuje jeho jednotlivé atributy v potřebném formátu pro úpravy. V tomto formátu jsou atributy rozšířené o přeložené názvy a informaci, zda-li byla jeho hodnota převedená do JSON řetězce, pomocí které se po odeslání formuláře pro uložení záznamu hodnota JSON řetězce převede zpátky na objekt. Po odeslání tohoto formuláře pro uložení se projde pole s modifikovanými atributy a dle názvů atributů se aktualizují hodnoty uvnitř první proměnné, která ukládá přijatá data v originálním formátu. Poté, co se aktualizují hodnoty uvnitř originální proměnné, odešle se do kontroléru pomocí metody PUT.

V kontroléru se celý proces ještě jednou zkomplikuje, jelikož spojení s Entras API zprostředkovává server. Musí se tedy projít celé tělo přijatého HTTP požadavku pomocí *BufferedReaderu*, vyčíst z jeho těla po částech zaslaný JSON a ten uložit do řetězce. Uložený řetězec se musí následně opět vložit do nového požadavku vytvořeného na servrové části aplikace pro Entras API a ten následně odeslat. Pokud dojde k chybě při aktualizaci uvnitř Entras API, tak je potřeba opět informovat uživatele o neúspěchu. V případě, že k chybě nedojde, tak se aktualizuje původní seznam zobrazený před vybráním editace a ten se následně opět zobrazí.

V následujícím výpise 7 je zobrazená funkce napsaná v jazyce Java, která umožňuje vytvořit PUT požadavek a vložit mu zadané řetězce jako adresu a tělo. Funkce poté požadavek odešle a ze získané odpovědi vyčte její tělo, které vrátí jako řetězec.

```
1 public String editProperty(String url, String jsonData) throws UnirestException {
2     HttpRequestWithBody putRequest = Unirest.put(url);
3     putRequest.body(jsonData);
4     return putRequest.asString().getBody();
5 }
```

Výpis 7: Vytvoření PUT požadavku v jazyce Java pomocí Unirest knihovny.

Jelikož vytváření HTTP požadavků v jazyce Java není úplně obvyklé, využívá aplikace knihovnu *Unirest*, která poskytuje odpovídající funkce umožňující jednoduchou tvorbu HTTP požadavků.

6 Závěr

V praktické části této práce vznikl nástroj pro rychlejší a snadnější správu Raspberry Pi a konfiguraci prostředků používaných v rámci projektu Entras. Rozhraní tohoto nástroje, které je tvořené webovou stránkou, umožní provádět většinu konfigurace v jednotném a uživatelsky přívětivějším prostředí, oproti provádění konfigurace skrze SSH pro Raspberry Pi a ve výchozím webovém rozhraní umístěném na převodnících.

Celý proces se usnadní také tím, že nebude nutné pamatovat si spoustu konfiguračních příkazů, znát syntaxi konfiguračních skriptů a vědět přesné nastavení pro převodníky.

V posledním případě poskytne aplikace také rozhraní pro přístup k NFC čtečkám a Entras API, čímž umožní nahlížet do prvků přístupového systému, a v případě potřeby provádět i zde základní správu.

Objektově orientovaný návrh této aplikace umožní její snadnou rozšiřitelnost a znovupoužitelnost jednotlivých modulů, ať už jde o balík určený pro práci se systémem, konfiguraci převodníků pomocí sítě nebo *ntp* a *dhcpcd* služby.

Práci hodnotím pozitivně, jelikož bude přínosem pro projekt Entras a zároveň umožnila i mě získat spoustu praktických znalostí zaměřených na technologie, jako jsou Spring, Angular a Maven pro tvorbu moderních webových aplikací. Mimo to hodnotím kladně i to, že jsem si mohl vyzkoušet vytvořit rozsáhlejší projekt.

Literatura

[1] FIELDING, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures*. University of California, Irvine, 2000.

[2] FREEMAN, Adam. *Pro Angular*. Second Edition. 2017. ISBN 9781484223079.

[3] ČÁPKA, David. *Úvod do Single Page Application v ASP.NET*. [online]. [cit. 2018-06-04]. Dostupné z: https://www.itnetwork.cz/csharp/asp-net/single-page-application/tutorial-uvod-do-asp-net-single-page-application

[4] WALLS, Craig. *Spring Boot in Action*. First Edition. 2016. ISBN 9781617292545.

[5] HORDĚJČUK, Vojta. *Maven*. [online]. [cit. 2017-01-09]. Dostupné z: http://voho.eu/wiki/maven/

[6] *Apache Tomcat*. [online]. [cit. 2018-16-04]. Dostupné z: http://tomcat.apache.org

[7] *NFC přístupový systém pro bytové domy*. [online]. [cit. 2018-18-04]. Dostupné z: http://www.entras.cz

[8] SHINDE, Mandar, */etc/shadow file format in Linux Explained*. [online]. [cit. 2015-02-08]. Dostupné z: http://www.yourownlinux.com/2015/08/etc-shadow-file-format-in-linux-explained.html

[9] *USR-TCP232-E45/M4 Config Protocol*. [online]. [cit. 2017-24-11]. Dostupné z: https://www.usriot.com/download/M4/E45&M4%20config%20protocol%20V1.4.pdf

[10] *Same-origin policy*. [online]. [cit. 2018-02-03]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy

[11] *System Applications*. [online]. [cit. 2015-03-09]. Dostupné z: https://www.w3.org/wiki/System_Applications

[12] NILSSON, Claes. *TCP and UDP Socket API*. [online]. [cit. 2015-23-07]. Dostupné z: https://www.w3.org/TR/raw-sockets/

[13] *B2G OS*. [online]. [cit. 2017-11-04]. Dostupné z: https://developer.mozilla.org/en-US/docs/Archive/B2G_OS

[14] *About HTML5 WebSocket*. [online]. [cit. 2018-19-04]. Dostupné z: https://websocket.org/aboutwebsocket.html

A Spuštění aplikace

Pro spuštění aplikace je vyžadován server Apache Tomcat, volitelně Maven pro sestavení back-end části aplikace napsané pomocí platformy Spring a Node.js pro sestavení front-end části aplikace napsané pomocí platformy Angular.

1. Pro sestavení front-end části je nutno stáhnout závislosti využívané platformou Angular, to lze provést zadáním příkazu *npm install --force* uvnitř adresáře *frontend/* a poté zadáním příkazu *ng build*, který sestaví front-end a umístí jej do adresáře back-end části (čímž se bude nasazovat front-end spolu s back-end částí).

2. Sestavení back-end části lze provést zadáním příkazu *mvn clean package [-DskipTests]* v kořenovém adresáři aplikace. Tím vznikne soubor *target/ROOT.war*, připravený pro nasazení na Tomcat (pro usnadnění by tento soubor měl již být uvnitř adresáře *target/*).

3. Tento .war soubor nyní stačí nasadit na Tomcat a aplikace bude dostupná skrze prohlížeč na adrese *http://localhost:8080* (port se může lišit podle konfigurace serveru Tomcat).

A.1 Požadavky pro plnou funkčnost

Pro plnou funkčnost je nutné splnit několik následujících požadavků. Mimo to je vyžadován Linux (ideálně Raspbian).

1. Aplikace vyžaduje oprávnění pro spouštění vlastních Bash skriptů jako sudo. Toho lze docílit dvěma způsoby. Prvním a jednodušším způsobem je přidání záznamu *tomcat8 ALL=(ALL) NOPASSWD: ALL* do souboru */etc/sudoers*, čímž získá uživatel *tomcat8* oprávnění pro spouštění všech skriptů jako sudo. Druhým a složitějším způsobem je uvedení pouze jednotlivých skriptů, které aplikace využívá, tak jak je to popsáno v kapitole 5. Tyto skripty jsou umístěné uvnitř adresáře aplikace, po nasazení na Tomcat tzn. v adresáři */var/lib/tomcat8/webapps/ROOT/WEB-INF/classes/bash*. Podle verze serveru Tomcat se může lišit cesta k adresáři s nasazenými webovými aplikacemi a také se může lišit název uživatele, například se může jmenovat *tomcat* místo *tomcat8*. Aby se změny v souboru */etc/sudoers* projevily, je nutné restartovat počítač.

2. Je nutné mít nainstalovanou službu *ntp*, tu lze nainstalovat na systému Raspbian zadáním příkazu *sudo apt-get install ntp* do terminálu.

3. Je nutné mít nainstalovanou službu *dhcpcd* ve verzi 6, tu lze nainstalovat na systému Raspbian zadáním příkazu *sudo apt-get install dhcpcd* do terminálu. Pro Raspbian 4.9+ by tato služba již měla být nainstalována spolu se systémem. Podle dostupných repozitářů se může lišit název a verze služby.

B Přílohy ve formě CD

Součástí práce je CD, které obsahuje zdrojové kódy aplikace vzniklé v rámci praktické části této práce, a dále teoretickou práci v elektronické podobě.